CS1010 Programming Methodology
Week 6: One-dimensional Arrays (Selected Answers)

> It is not hard to learn more. What is hard is to unlearn when you discover yourself wrong. ~ *Martin H. Fischer*

III. Arrays

Important notes:

As we are basing on ANSI C (or C90), we do NOT allow variable-length arrays. Hence the following code is not permitted, because at compilation time the system wouldn't know what value **n** contains and hence the size of the array is unknown:

```
int n, a[n];
printf("Enter n: ");
scanf("%d", &n);
. . .
```

However, as gcc is C99-compliant (C99 is a newer standard for C, which permits variable-length arrays), the code above can be compiled without warning, even if –Wall option is used.

You may compile with the –pedantic option instead, in which case a warning message will be issued.

For all problems on arrays, we will specify the maximum size of an array, so that you can declare the array with the right size. We will not accept the use of variable-length arrays.

4. **Logical thinking.**
   An array is a collect of data. It is very common to ask the following two questions about a collection: (a) do all the data in the collection share a certain property? (b) does there exist one datum that has a certain property? The former is a *universal* question, and the later an *existential* question.

   For example, "are all the values in the array non-negative?" is a universal question; "is there one value in the array that is negative?" is an existential question. In this case, the two questions are actually the same, hence we can transform one into another.

   Write a function **nonNegative(int arr[], int size)** that returns 1 (true) if all the elements in arr[0]… arr[size-1] are non-negative; or returns 0 (false) otherwise. You may assume that the array has at least one element.

Answer: See **q4_ans.c**

```
// Return 1 if all elements are non-negative; otherwise 0
// Some students may use while loop, or break statement.
// Precond: size > 0
int nonNegative(int arr[], int size) {
   int i;

   for (i=0; i<size; i++)
      if (arr[i] < 0)
         return 0;

   return 1;
}
```

A common mistake students make is hasty conclusion: the moment one non-negative element is encountered, the program concludes that the whole array is non-negative. The other case is more of an efficiency issue: a negative element is encountered, but the program goes on to check the rest of the array, which is unnecessary.

5. **Logical thinking.**
Given an array of integers, write a function **isSorted(int arr[], size)** that returns 1 (true) if the array **arr** is sorted in non-decreasing order, or returns 0 (false) otherwise. You may assume that the array has at least one element.

For example, 3, 12, 15, 18 and -5, 8, 8, 10 are in non-decreasing order, but 4, 6, 9, 7, 12 are not.

Do you see any similarity between this question and Q4? (Yes, Computational Thinking: **Pattern Recognition**!) For this question, what is the property you have "abstracted" out to check?

Answer: See **q5_ans.c**

The property is that for the array to be sorted in non-decreasing order, every element must be at least as big as its immediate previous element. This leads to checking each element (starting from the second element) with its immediately previous element.

```
// Return 1 if all elements are non-negative; otherwise 0
// Some students may use while loop, or break statement.
// Precond: size >0
int isSorted(int arr[], int size) {
   int i;

   for (i=1; i<size; i++)
      if (arr[i] < arr[i-1])
         return 0;

   return 1;
}
```

6. **Checking duplicates.**

   Write a program **duplicates.c** to fill an integer array with *n* ($1 \le n \le 1000$) random integers whose values are in the range [*lower*, *upper*] where *n*, *lower* and *upper* are inputs from user. Moreover, $0 < lower < upper$. Your program then computes the total number of duplicates in the array.

   For example, if a 15-element array contains the values { 97, 12, 45, 97, 23, 12, 53, 97, 30, 30, 10, 53, 8, 1, 53 }, then there are altogether 8 duplicates (three 97s, two 12s, and three 53s).

   (We will revisit this question after we have covered sorting. For the moment, do not use sorting in your solution.)

   Answer: See **q6_ans.c**